

并发三大特性: ①可见性
②原子性
③有序性. 指令按照预期顺序执行



volatile 定义——保证共享变量的一致性。前面加了Lock.

原理——有volatile的共享变量写会多出一行汇编代码。

lock前缓存在多核CPU下:

- ① 将当前 CPU 缓存行写回内存。
- ② 写回内存的操作会使其他CPU里缓存了该内存地址的数据失效。

缓存一致性协议. 每个CPU通过总线上传播的数据来检查自己缓存是否过期。

volatile { 可见性. 禁止重排 } 读: 禁止重排之后. 写: 禁止重排之前. volatile 标记: 解决编译器层面. 内存屏障: 解决硬件层面 } 的可见性与重排序问题.

使用优化.

① 采用追加字节的方式优化出队和入队. 避免头节点和尾结点 load 到同一缓存行. Jdk7. J.U.C 中的 Linked Transfer Queue 追加到了 64 字节.

原理: CPU 高速缓存行是 64 字节.

若一个 node 不足 64 字节. → 不同节点会读入同一个缓存行.

→ CPU 试图修改一个 node → 锁住整个缓存行.

→ 缓存一致性机制触发 → 其他 CPU 无法访问自己缓存行 node.

适用场景 ① 缓存行为 64B 的 CPU.
② 共享变量会被频繁地写.

synchronized.

① 原理.

基础: What: Java 中每个对象都可以作为锁.

形式 { 普通同步方法. 一锁: 当前实例对象.
静态同步方法. 一锁: 当前类的 Class 对象.
同步方法块. 一锁: () 内的对象. 实现: monitoreenter, monitorexit.

who:

同步方法块。—— 锁：()内的对象。—— 实现：monitorenter、monitorexit。

where:

Java对象头 { Mark Word, Class Metadata Address, Array Length. 数组类型独有
hashcode, 分代年龄, 锁标记位, 到对象类型数据的指针, 数组长度.

② 锁升级和对比.

重量级锁.
轻量级锁.
偏向锁状态.
无锁状态.

锁只可升不可降.

↑ 提高获得锁和释放锁的效率.

① 偏向锁.

对象头中存储 1 bit 表示: 是否为偏向锁.

是 采用 CAS 将对象头的偏向锁指向当前线程.
否 走 CAS 竞争锁.

(1) 偏向锁的撤销.

等到竞争出现才释放锁的机制.

(2) 关闭偏向锁

-XX:-UseBiasedLocking=false.

应用场景: 锁通常情况下属于竞争状态.

② 轻量级锁.

(1) 加锁.

先 CAS, 失败则用自旋锁.

(2) 解锁.

表2-6 锁的优缺点的对比

锁	优点	缺点	适用场景
偏向锁	加锁和解锁不需要额外的消耗, 和执行非同步方法相比仅存在纳秒级的差距	如果线程间存在锁竞争, 会带来额外的锁撤销的消耗	适用于只有一个线程访问同步块场景
轻量级锁	竞争的线程不会阻塞, 提高了程序的响应速度	如果始终得不到锁竞争的线程, 使用自旋会消耗 CPU	追求响应时间 同步块执行速度非常快
重量级锁	线程竞争不使用自旋, 不会消耗 CPU	线程阻塞, 响应时间缓慢	追求吞吐量 同步块执行速度较长

原子操作的实现原理.

术语:

缓存行 Cache Line.

缓存的最小操作单位.

比较并交换 CAS

→ 每步需要一个时钟周期.

CPU 流水线 CPU pipeline.

指令分解为 n 步, 由 CPU 不同单元流水线执行, 实现一个时钟周期执行完一个指令.

内存顺序冲突 Memory order violation.

由假共享引起. 发生时 CPU 必须清空流水线.

多个 CPU 同时修改同个 Cache Line → 其中一个 CPU 操作无效.

② CPU 如何实现原子操作.

内存操作 { 简单 CPU 自动保证.
跨总线宽度.

内存操作 { 简单 CPU自动保证.
复杂 ① 何为复杂? { 跨总线宽度.
跨多个缓存行.
跨页表.
② 如何保证原子性 { 总线锁定.
缓存锁定.

1) 总线锁 Lock #信号. volatile 原理.
但一般由缓存锁代替. 因为它把CPU和内存之间的通信锁住了.

2) 缓存锁.
保证某个内存地址的操作是锁定的.

不会用缓存锁的情况 { CPU不支持.
操作的缓存不能被缓存在CPU内部.
操作的数据跨多个缓存行.

③ Java如何实现原子操作. { 锁
循环CAS. 本质为CPU的CMPXCHG指令.